社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

大規模グラフに対する高速クリーク列挙アルゴリズム

宇野　毅明†

† 国立情報学研究所, 〒 101-8430 東京都千代田区一ツ橋 2-1-2
E-mail: †uno@nii.jp

あらまし　グラフ $G = (V, E)$ の頂点集合 S の任意の頂点間に枝があるものをクリークとよぶ. また, 2 部グラフ $G = (V_1 \cup V_2, E)$ の頂点集合 S に対して, S の任意の頂点 $v_1 \in V_1$ と $v_2 \in V_2$ に対して枝があるものを 2 部クリークとよぶ. 本稿では, 巨大で疎なグラフ極大クリークと極大 2 部クリークを列挙する, 実用的に高速なアルゴリズムを提案する. グラフの最大次数を Δ とすると, 本稿の改良により, 極大クリーク 1 つあたりの計算時間が $O(|V||E|)$ から $O(\Delta^4)$ に, 極大 2 部クリークは $O(|V||E|)$ から $O(\Delta^3)$ に改善された. 計算機実験により, ある程度ランダムな入力に対しては, 1 つあたり $O(\Delta^2)$ 時間程度しかかからないことを示し、さらに現実のデータに対しても高速であることを示す.

キーワード　数え上げ、発生、計算量、ならし解析、疎グラフ

Fast Algorithms for Enumerating Cliques in Huge Graphs

Takeaki UNO†

† National Institute of Informatics, 2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo, JAPAN, 101-8430
E-mail: †uno@nii.jp

Abstract　A vertex subset of a graph $G = (V, E)$ is called a clique if any two vertices of S are connected by an edge. A vertex subset of a bipartite graph $G = (V_1 \cup V_2, E)$ is called a bipartite clique if any two vertices $v_1 \in V_1$ and $v_2 \in V_2$ are connected by an edge. In this paper, we propose a practical fast enumeration algorithm for maximal cliques and bipartite maximal cliques of huge sparse graphs. The time complexity per maximal clique is reduced from $O(|V||E|)$ to $O(\Delta^4)$, and that of maximal bipartite cliques is reduced from $O(|V||E|)$ to $O(\Delta^3)$. By computational experiments, we show that the algorithm takes $O(\Delta^2)$ in random instances, and the computation time is very short for some real world problems.

Key words　generation, listing, amortized analysis, complexity, sparse graph

1. はじめに

グラフ $G = (V, E)$ の頂点集合 S で, 任意の 2 つの頂点の間に枝があるものを G の**クリーク**とよぶ. G が 2 つの頂点集合 V_1, V_2 により構成される 2 部グラフである場合, S の任意の 2 つの頂点 $v_1 \in V_1, v_2 \in V_2$ に対して $(v_1, v_2) \in E$ であれば S を G の**2 部クリーク**とよぶ. 集合の意味で極大なクリークを極大クリーク（極大 2 部クリーク）とよぶ. クリークは, グラフ論の中でも基礎的なものであり, 最適化問題を含め多くの研究がなされている.

クリークは, 他の分野での応用も多い. データマイニングやネットワーク・計算言語学モデルの分野では, データセットの対応, コンピューターネットワーク, 異種言語の意味的な対応を表すグラフなどから極大クリークを見つけ出し, 対象となるデータを分析しようという試みが多く行われてきている. クリークは, 相関の大きいデータの集合を表すなど, 自然なモデル化に基づいて導き出されるものだからである.

最近のデータマイニングやネットワーク・言語学モデル, グラフマイニングなどの分野では, 列挙を行う必要性が高まっている [1], [2], [4]. 例えばグラフマイニングでの, 入力されたグラフの中で頻出する部分グラフを求める問題などは, 本質的に列挙を行うことにより求解を行うアルゴリズムが多い. しかし, 列挙自体に時間がかかるようでは, 列挙という手法を用いるモチベーションが低くなる. 例えばウェブネットワークの中からコミュニティーを見つけ出す [4] の研究では, 非常に大規模なグラフの極大クリークを列挙に挑戦しているが, 列挙に時間がかか

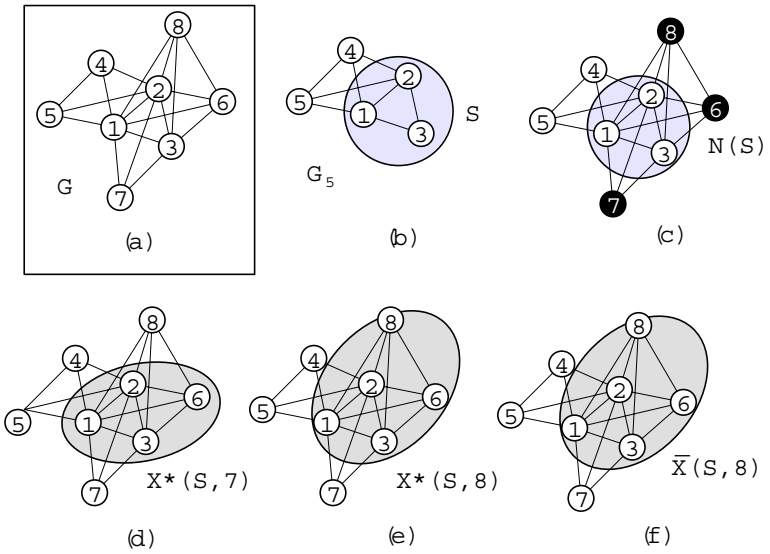


図 1　(a) グラフ G, (b) グラフ G_5 と 5 極大クリーク S, (c) 黒塗りの頂点が $N(S)$, (d) $X^*(S,7)$, (e) $X^*(S,8)$, (f) $\bar{X}(S,5)$

りすぎるため, モデルを簡易化することで, 擬似的に問題を解いている. これら大規模なデータは, 疎であることが多い. そこで, 疎であることを上手に高速アルゴリズムを開発すれば, これらの研究を前進させることにつながるであろう.

グラフ G のクリークは, G の補グラフでは安定集合になる. 安定集合とは, 任意の 2 つの頂点間に枝がないような頂点集合のことである. 極大安定集合の列挙問題に対しては, 1977 年に Tsukiyama, Ide, Ariyoshi, Shirakawa [5] によって出力線形時間アルゴリズムが提案されている. 出力線形時間であるとは, 計算時間が出力数に比例するということである. このアルゴリズムの時間計算量は, 出力される極大安定集合 1 つあたり $O(|V||E|)$ である. このアルゴリズムを用いれば, 極大クリークも 1 つあたり $O(|V||E|)$ 時間で列挙できが, グラフが巨大になると, 計算時間は極端に大きくなる. 本研究では, このアルゴリズムに, グラフが疎であることに基づいた改良を加えることにより, 時間計算量を解 1 つあたり, 極大クリークでは $O(\Delta^4)$, 極大 2 部クリークでは $O(\Delta^3)$ に下げた. これにより, 解 1 つあたりの計算時間は頂点数・枝数に依存しなくなり, 大規模ネットワークに対する大幅な計算時間の減少が期待される. さらに, 実装上の工夫やグラフの次数のばらつきに対応するための工夫を導入した. これらの結果, 計算実験により, 平均的な計算時間は $O(\Delta^2)$ 程度であることを実証した.

2. 基本アルゴリズム

この節では, [5] のアルゴリズムを簡単に説明する. まず入力グラフであるが, グラフが疎である場合を主に考えるので, グラフは頂点の隣接リストで表現されるとする. つまり各頂点について, その頂点に隣接する頂点のリストがあるとする. $V = \{v_1, ..., v_n\}$ とし, G に対して G_i を頂点集合 $\{v_1, ..., v_i\}$ によって誘導される部分グラフとする. 頂点集合 S がグラフ G_i の極大クリークであるとき, S を i 極大クリークとよぶ. [5] のアルゴリズムは, 1 極大クリークから $|V|$ 極大クリークまでのすべての i 極大クリークを列挙するものである. 頂点集合 S に対し, S の全ての頂点と隣接する頂点の集合を $N(S)$ とする. また, クリーク S に対して,

・ S に加えても S がクリークであるような頂点の中で, 最小添え字な頂点を, 添え字が i 以下であれば, S に加える

という作業を S が極大になるまで繰り返して得られる頂点集合を $X^*(i,S)$ と表記する. $X^*(i,S)$ は, $N(S)$ の最小添え字頂点を S に追加し、$N(S)$ を更新する、という作業を $N(S)$ の頂点の添え字が全て $i+1$ 以上になるまで繰り返して得られるものと同じである。S に頂点 v を追加したときの $N(S)$ の更新は、$N(S)$ から v に隣接しない頂点を取り除くことに行える。これは $O(\Delta)$ 時間で行えるので、S から $X^*(i,S)$ を求める作業は $O(|X^*(i,S)|\Delta)$ 時間で行える。以上の用語の定義は, 図 1 を参照されたい.

i 極大クリーク S に対して, S の親 $P(S,i)$ を以下の $i-1$ 極大クリークで定義する.

$$P(S,i) = \begin{cases} S & \text{if} \quad v_i \notin S \\ X^*(i-1, S \setminus \{v_i\}) & \text{if} \quad v_i \in S \end{cases}$$

1 極大クリーク $\{v_1\}$ 以外の全ての i 極大クリークに唯 1 つ親が定義され, かつ, i 極大安定集合の親は $i-1$ 極大安定集合であることから, この親子関係をグラフで表現すると, $\{v_1\}$ を根とする根付き木 (列挙木とよぶ) になる. 図**??** に 6 頂点のグラフの列挙木を示した. この木を深さ優先探索すれば, 全ての i 極大クリークを列挙することができる. 探索をするときに, この木全体をメモリに蓄える必要はない. i 極大クリークの子供

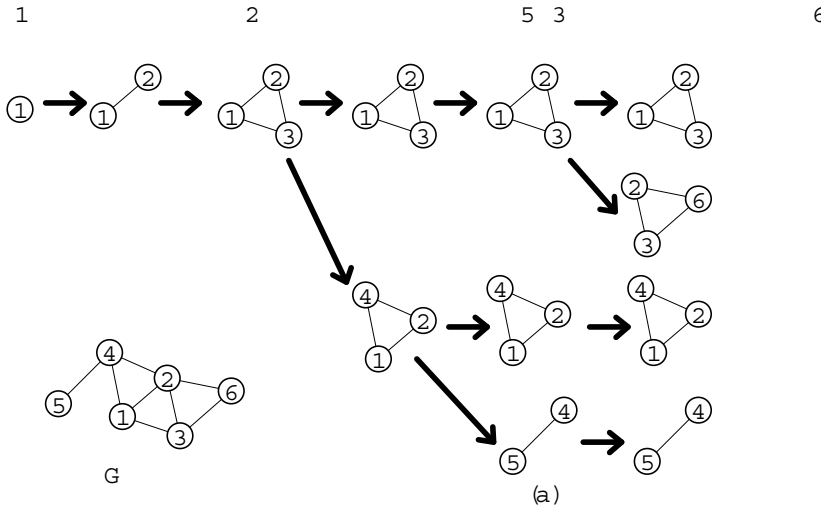


図 2　グラフ G のクリークが作る列挙木. 横向きの矢印が type1 子供、斜めの矢印が type2 子供を指す. 頂点 6 は (a) のクリークに隣接しないので, (a) での type2 子供のチェックは省略できる.

を見つけるアルゴリズムを用いれば, 現在訪れている頂点の子供に対して, 再帰呼び出しをする, という方法で, メモリを浪費しない探索が行える.

$\bar{X}(S,v)$ を S に v を加え, v に隣接する頂点をすべて取り除いて得られる頂点集合とする. S の子供 S' に対して,

$$S' = \begin{cases} S & \text{if} \quad v_i \notin S' \\ \bar{X}(S, v_{i+1}) & \text{if} \quad v_i \in S' \end{cases}$$

が成り立つ. よって, S の子供は S 自身か $\bar{X}(S', v_{i+1})$ の高々 2 つである.

もし $S' = S \cup \{v_{i+1}\}$ が $i+1$ 極大クリークであれば, $P(S', i+1) = S$ であるので, S' は S の子供である. もし $S \cup \{v_{i+1}\}$ がクリークでなければ, $S' = S$ は $i+1$ 極大クリークである. $P(S', i+1) = S$ となるので, S' は S の子供になる. このようにして得られる S' を S の type 1 の子供とよぶ. 任意の i 極大クリークは, $i < |V|$ であれば, 必ず type 1 の子供を持つ. $S \cup \{v_{i+1}\}$ がクリークでないとき, $\bar{X}(S, v_{i+1})$ は S の子供になる可能性がある。もし $\bar{X}(S, v_{i+1})$ が $i+1$ 極大クリークであり, かつ $P(\bar{X}(S, v_{i+1}), i+1) = S$ であれば, またそのときに限り $\bar{X}(S, v_{i+1})$ は S の子供になる. これを S の type 2 の子供とよぶ. $\bar{X}(S, v_{i+1}), X^*(i-1, S \setminus \{v_i\})$ ともに $\mathrm{O}(|S|\Delta)$ 時間で求められるで, $|S| \leqq \Delta$ より, S の子供は $\mathrm{O}(\Delta^2)$ 時間で求められる.

任意の i 極大クリークから type 1 の子供を最低 $|V|$ 回たどると G の極大クリークに到達する. よって, 極大クリーク 1 つあたりの計算時間は $|V|$ 回の反復の計算時間となり, $\mathrm{O}(|V|\Delta^2)$ となる. もう少し詳しく解析すると, 極大クリーク 1 つあたりの計算時間は $\mathrm{O}(|V||E|)$ となる.

3. 改良アルゴリズム

本研究での改良の基本的な方針は, type 2 子供が発生しない反復を省略することである. i 極大クリーク S に対して, v_{i+1} が S のどの頂点にも隣接していないとする. この場合, S が type2 子供を持てば, それは $\bar{X}(S, v_{i+1}) = \{v_{i+1}\}$ である. これら 1 頂点のみからなる i 極大クリークのみを別口で発生させれば, v_{i+1} が S に隣接しない場合は type 2 子供の発生が不要になる。i 極大クリーク S に対して, type 1 の子供を再帰的に生成して得られる極大クリーク S' は $X^*(|V|, S)$ であり, これは . $\mathrm{O}(|S'|\Delta)$ 時間で構築できる。S から $X^*(|V|, S)$ を得るまでに通過する反復のうち、type2 子供の存在を調べる必要があるものは、その反復で追加する頂点 v_k に, 頂点 $v_j \in X^*(|V|, S), j > k$ が隣接するときである。これらの頂点を添え字順に並べたリストを $J(S, i)$ とする。$|J(S, I)|$ $=\mathrm{O}(\Delta^2)$ であるので、S から $X^*(|V|, S)$ を得るまでの反復の計算時間は $\mathrm{O}(\Delta^4)$ となる。

改良アルゴリズムは以下のように記述できる. ただし, 頂点集合は添え字順にソートされたリストで保持するものとし, リスト L の末尾の頂点を $l(L)$ と表記する. L が空の場合 $l(L)$ には v_{-1} という架空の頂点が入るものとする.

Procedure Enum_Maximal_Clique_iter
(i 反復の深さ, S: i 極大クリーク)
(EMC1) $J := J(S, i), S := X^*(|V|, S)$ とし,
$N(S)$ を更新し, S を出力する
(EMC2) $v_j := l(J), v_h := l(S)$ とする
(EMC3) **If** $j, h \leqq i$ **then return**
(EMC4) **If** $h > j$ **then** $S := S \setminus \{v_h\}$
(EMC5) **Else**

(EMC6) $J := J \setminus \{v_j\}, S' := \bar{X}(S, v_{i+1})$
(EMC7) **If** $N(S') = \emptyset$ **then**
(EMC8) $S' := X^*(i, S' \setminus \{v_j\})$
(EMC9) **If** $S' = S$ **then**
ENUM_MAXIMAL_CLIQUE_ITER(S', j)
(ECM10) **End if**
(EMC10) **End if**
(EMC11) **End if**
(EMC12) **Go to** (EMC2)

Procedure ENUM_MAXIMAL_CLIQUE_MAIN($G = (V, E)$)
(EMCM1) **For** $i := 1$ **to** $|V|$
(EMCM2) **If** v_i に隣接する任意の頂点の添え字が i 以上
then ENUM_MAXIMAL_CLIQUE_ITER($\{v_i\}, i$)
(EMCM3) **End for**

4. 2 部極大クリーク列挙の高速化

この節では, グラフ $G = (V = V_1 \cup V_2, E)$ の極大 2 部クリークを列挙するアルゴリズムの改良を説明する. 簡単のため, V_1 のすべての頂点に隣接する V_2 の頂点, および V_2 のすべての頂点に隣接する V_1 の頂点は存在しないとする. 以下では, 頂点の添え字は V_1 の頂点が $1, ..., |V_1|$ であり, V_2 の頂点が $|V_1 + 1, ..., |V_1| + |V_2|$ という条件を満たすとする. この条件は本質的であり, この条件が満たされていないと成り立たない議論が存在することを注意しておく. また, 以下では特に断らない限り $i > |V_1|$ であるとする.

グラフ G の 2 部クリークは, V_1 と V_2 がクリークになるように枝を追加したグラフ, つまり $(V_1 \cup V_2, E \cup V_1 \times V_1 \cup V_2 \times V_2)$ のクリークと対応する. よって, G の極大 2 部クリークはこのグラフの極大クリークの列挙により列挙できる. しかし, この方法は今回の研究対象のような疎なグラフに対しては歓迎されない. 枝の追加によりグラフが極めて密になるからである.

このような枝の追加を, 実際に実際に行わず, 仮想的に行ったとしても計算時間は短縮されない. なぜならば, 上記アルゴリズムの計算時間は最大次数に依存し, このグラフの最大次数は $|V_1|, |V_2|$ より大きくなるからである. これでは速度の向上は見込めない.

さらにもう 1 つ問題がある. それは, 任意の部分グラフ G_i に対して, $i \leqq |V_1|$ であれば $\{1, ..., i\}, i > |V_1|$ であれば $\{1, ..., |V_1|\}$ と $\{|V_1| + 1, ..., i\}$ は, それぞれ i 極大 2 部クリークになるということである. これらを自明なクリークとよぶ. 自明なクリークは頂点数が大きく, 自明なクリークを扱う反復は, 多大な時間計算を要する.

そこで本研究では, 上記一般のクリーク用の改良を 2 部クリーク用に焼きなおし, 自明なクリークの列挙を省略する方法を提案する. 頂点集合 S に対して, $S_1 = S \cap V_1, S_2 = S \cap V_2$ とする. i 極大 2 部クリーク S に対して,

$$N(S_1) = S_2, N(S_2) = S_1$$

が成り立つことを注意しておく. 特に断らない限り, i 極大 2 部クリークは自明でないクリークを指し, $i > |V_1|$ であるとする.

i 極大 2 部クリークを G_i の極大 2 部クリークで定義する. 任意の i 極大 2 部クリーク S に対して, $X^*(i, S)$ と $P(S, i)$ を i 極大クリークに対する定義と同じ定義で定める.

$$X^*(i, S) = S \cup N(S_2) \cup N(S_1 \cup N(S_2)) \cap \{1, ..., i\}$$

であることを注意しておく. i 極大クリークと同様, i 極大 2 部クリーク S は type 1 の子供を必ず 1 人と高々1 人の type 2 子供を持つ. この親子関係は $|V_1|$ 極大 2 部クリーク V_1 を根とする木を構成する.

i 極大 2 部クリーク S に対して, $\bar{X}(S, v_{i+1})$ を S から v に隣接しない S_1 の頂点を取り除き, v を加えて得られる 2 部クリークとする. S' が S の type 2 の子供であれば, $S' = \bar{X}(S, v_{i+1})$ が成り立つ. $N(\bar{X}(S, v_{i+1}) \setminus \{v_{i+1}\}) = S_1$ であることから,

$$X^*(i, \bar{X}(S, v_{i+1}) \setminus \{v_{i+1}\}) = S$$

が必ず成り立つ. よって, $\bar{X}(S, v_{i+1})$ が $i + 1$ 極大クリークであれば, またそのときに限り, $\bar{X}(S, v_{i+1})$ は S の type 2 子供となる.

自明でない i 極大 2 部クリーク S の親 $P(S, i)$ が自明なクリークであるとき、S は V_2 の頂点をちょうど 1 つ含む. これら自明なクリークを親に持つクリークを別口で発生させれば, 自明なクリークは探索せずともすべての自明でないクリークを列挙できる。また, i 極大 2 部クリーク S に対し、$V_1 \cap S$ の頂点で v_{i+1} に隣接するものが 1 つしかない場合、その type 2 子供は V_1 の頂点を 1 つしか含まない。これらも別口で発生させると、結局、v_{i+1} に隣接する頂点が 2 つ以上あるときのみ、type2 の子供の存在を調べれば良い、ということになる。

以上のアイディアを用いると, 極大クリーク列挙のアルゴリズムとほぼ同じく, $X^*(|V|, S)$ と $J(S, i)$ を用いて反復を省略する, 極大 2 部クリーク列挙アルゴリズムが構築できる.

このアルゴリズムの 1 反復のボトルネックは, 各 $J(S, i)$ の頂点について type 2 子供の存在性を調べる部分であり, その計算時間は, $v \in J(S, i)$ に対して,

$$\mathrm{O}((v \text{ に隣接する } S \text{ の頂点数}) \times \Delta)$$

である. $v \in J(S, i)$ の全ての頂点についてこれの合計をとると, $\mathrm{O}(S$ の頂点に接続する枝数) になる. よって, 1 反復での type 2 の子供に関する計算時間の合計は $\mathrm{O}(|S|\Delta^2)$ となり, このアルゴリズムの計算時間は, 極大 2 部クリーク 1 つあたり $\mathrm{O}(\Delta^3)$ となる.

5. 次数にばらつきがある場合

ウェブネットワークなどでは, ほとんどの頂点の次数は定数とみなすことができるが, 一部の, 検索やリンクに関わるページに対する頂点のみ, 次数が極端に大きい. このようなグラフが入力として与えられると, 上記のアルゴリズムは次数の大きな頂点に対する計算時間が増大する. しかしこれは, いくつかの改良を施すことにより, 回避することができる. その改良を以下に示す.

まず、ある程度次数の大きな頂点 (例えば次数が $|V|/100$ 以上) に対してのみ、隣接行列を用意する。これにより、これら次数の大きな頂点に対しては、他頂点と隣接するかどうかを定数時間で判定できる。本稿のアルゴリズムは「v_i が S の全ての頂点に隣接するか」を頻繁に調べる。これには、v_i の次数が大きいと隣接リストを利用した場合の計算時間が長くなる。しかし v_i に関する隣接行列があれば、計算時間を $O(|S|)$ にできる。

さらに、頂点の添え字を次数の昇順に付け直す、という作業を行う。これにより、$|J(S,i)|$ の大きさを小さめに抑えることができる。type 2 子供候補の生成にかかる時間も S に含まれる頂点の次数に依存するので、ソートにより S に含まれる頂点の次数が平均的に小さくなるので、高速化が期待される。

この 2 つの改良により、極大クリーク 1 つあたりの計算時間は、平均次数にのみ依存するようになると見込まれる。特に, 定数 z に対して, 添え字 $|v|-z$ 以下の頂点の次数が $\bar{\Delta}$ である場合は、$i < |V|-z$ に対して

$$|J(S,i)| \leqq (\bar{\Delta} + z$$

となる。

アルゴリズムの実行中, 添え字が $|V|-z$ 以上の頂点のみからなる i 極大クリークの探索は打ち切り、それらは添え字が $|V|-z$ 以上の頂点が誘導するグラフを入力として別口で列挙すれば、極大クリーク 1 つあたりの計算量を $O(\bar{\Delta}^4)$, 2 部クリークの場合は $O(\bar{\Delta}^3)$ に下げることができる。

6. 計算実験と考察

本節では, 上記の 2 つのアルゴリズムを実際に計算機上に実装し, 計算実験を行った結果を報告する. 両者ともに C を用いて実装し, Pentium III 500MHz の PC で実験した. OS は Linux, コンパイラは gcc である. 今回の実験では, 実験に用いたグラフはランダムグラフである. ただし, 通常のランダムグラフを用いると, 大規模かつ疎なグラフでは極大クリークの大きさが極端に小さくなり, ほとんどの極大クリークが枝になってしまう. これでは実験の意味がないので, 一部が密で大部分が疎であるようなグラフを作成するべく, 以下の方法を用いた. 頂点の添え字は 0 から $|V|-1$ であるとする. r はパラメータである.

・各頂点 i に対し, 頂点 $(i-r)+|V| \bmod |V|$ から
$i+r \bmod |V|$ との間に 1/2 の確率で枝を張る

頂点を円周上に添え字順に並べ, 左右 r 個以内の近傍のみに 1/2 の確率で枝を作る, としたものと同じである. このグラフは局所的に密となるため, 極大クリークのサイズはそれなりに大きくなり, また, 頂点数が増加しても局所的な状況は変化しないため, 頂点数の増加に対してほぼ線形に極大クリーク数が増加する. 2 部グラフも同様にして発生させた.

・各頂点 $i \in V_1$ に対し, V_2 の頂点 $i-r+|V_2| mod |V_2|$ から
$i+r+|V_2| mod |V_2|$ との間に 1/2 の確率で枝を張る

実験では, $r=10, r=30$ の場合について, 頂点集合の大きさを変化させてグラフを生成し, 既存のアルゴリズムと本稿で提案したアルゴリズムを比較した (実験 1). また, 頂点数を 10000 に固定し, r を 10 から 300 まで変化させて生成したグラフに対して, 本稿のアルゴリズムを実行した (実験 2).

また, 一部だけ次数の大きいグラフに対する挙動を見るため, このグラフの一部の次数を大きくしたグラフについても実験を行った. $r=10$ で生成した一般グラフに対し, 頂点 $v_{|V|-40}, ..., v_{|V|}$ に対し, 頂点の次数を $|V|/2$ 程度になるよう, 枝をランダムに加えた. この結果, 頂点 $v_1, ..., v_{|V|-41}$ の次数は平均的に 30 程度になる. 頂点数を 1000 から 256000 まで変化させて生成したグラフに対して計算実験を行った (実験 3).

実験 1, 2, 3 の結果は以下の通りである. 計算時間は, 出力した極大クリーク 1 万個あたりの秒数である. なお, 既存のアルゴリズムについては, 計算時間が莫大になるものについては, 省略, あるいは極大クリークを 10000 個, あるいは 2000 個出力した時点で終了する, という処理を行っていることを注意しておく.

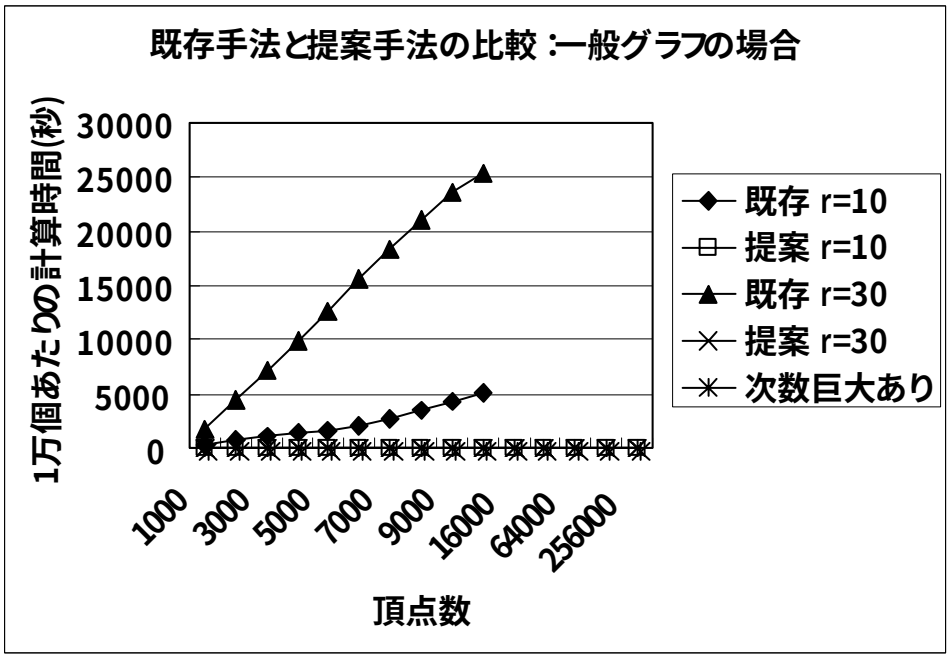
また、現実問題から得たデータに対する実験結果を実験 4 に示す。問題 1 は計算言語学の問題、問題 2 はデータマイニングで使われている問題であり、両者ともに 2 部グラフであり、次数にばらつきがある。

実験 1：一般グラフ, $r = 10, 30$

頂点数	1000	2000	4000	8000	10000	16000	32000	64000	128000	256000
旧 $r = 10$	378	761	1410	3564	5123					
旧 $r = 30$	1755	4478	9912	21085	25345					
新 $r = 10$	0.64	0.65	0.72	0.73	0.72	0.74	0.75	0.81	0.82	0.82
新 $r = 30$	4.41	4.44	4.47	4.56	4.51	4.54	4.55	4.91	4.88	4.88
解数 $r = 10$	2774	5553	11058	22133	27624	44398	89120	179012	357657	716978
解数 $r = 30$	20571	41394	83146	168049	209594	336870	675229	1352210	2711564	5411519

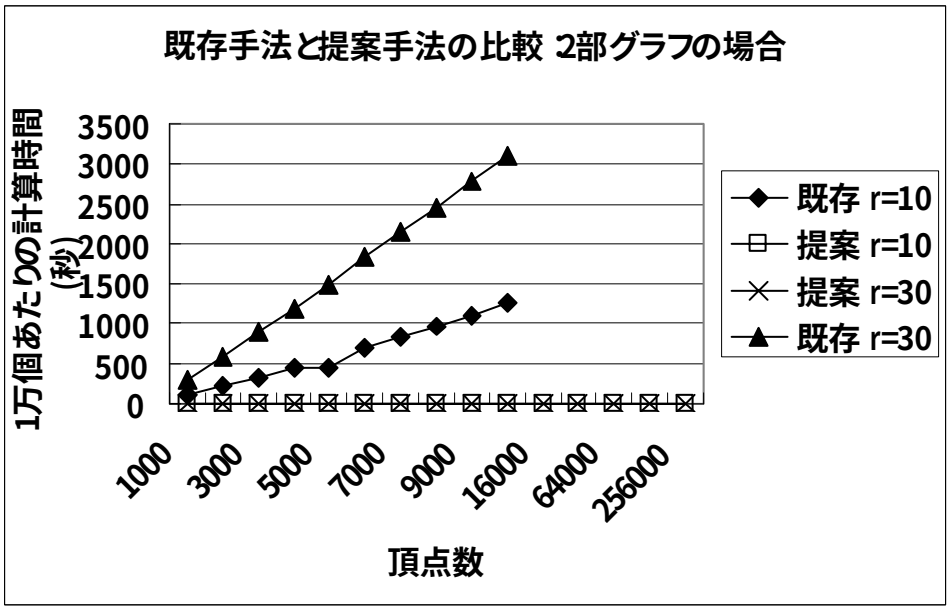
実験 3：一部の頂点の次数を大きくしたグラフ, $r = 10$

頂点数	1000	2000	4000	8000	10000	16000	32000	64000	128000	256000
新	1.07	1.14	1.12	1.31	1.21	1.36	1.74	2.62	4.02	7.8
解数	9136	18488	40427	68597	101697	165561	322149	625385	1233989	2307135



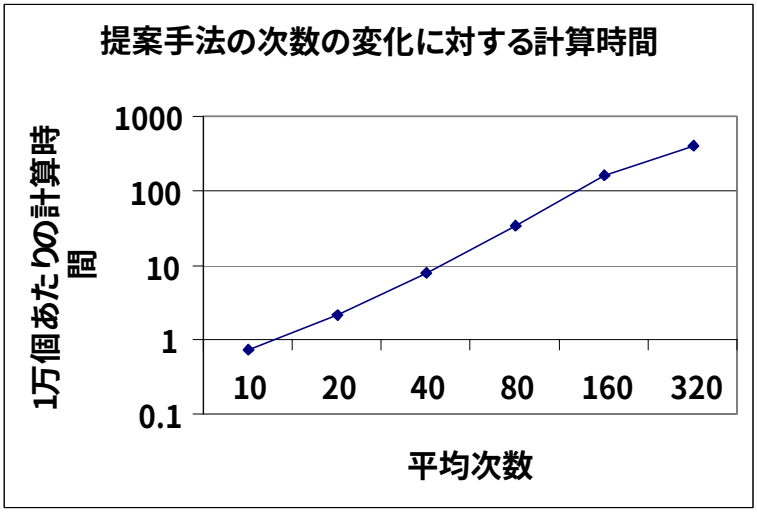
実験 1：2 部グラフ, $r = 10, 30$

頂点数	1000	2000	4000	8000	10000	16000	32000	64000	128000	256000
旧 $r = 10$	104	214	446	966	1260					
旧 $r = 30$	282	582	1190	2455	3100					
新 $r = 10$	0.33	0.32	0.3	0.3	0.27	0.3	0.3	0.34	0.34	0.35
新 $r = 30$	1.08	1.08	1.09	1.1	1.09	1.11	1.12	1.22	1.22	1.26
解数 $r = 10$	2085	4126	8316	16609	20862	33586	67143	134911	270770	541035
解数 $r = 30$	9136	18488	40427	68597	101697	165561	322149	625385	1233989	8351277



実験 2：一般グラフ, 頂点数 10000

r	10	20	40	80	160	320
新	0.72	2.11	7.93	34.4	159	394



実験 4: 現実の問題に適用した結果

	頂点数 (V_1, V_2)	枝数	極大 2 部クリーク数	計算時間	計算時間 (4 節の改良版)
問題 1	22677,18484	247003	2700737	104792 秒	1497 秒
問題 2	59602,497	149639	1427226	24534 秒	636 秒

実験 1 の結果より, 一般のグラフ, 2 部グラフどちらの場合も, 既存のアルゴリズム（旧と書かれたもの）の解 1 つあたりの計算時間は入力グラフの頂点数にほぼ比例するのに対して, 提案したアルゴリズム（新と書かれたもの）の解 1 つあたりの計算時間は, 最大次数さえ等しければ頂点数には依存しないことがわかる. また, 最大次数が 60 程度であれば, 提案したアルゴリズムの計算時間は, 解 1 つあたり 1/1000 秒以下であり, 実用的にも高速であることがわかる. 頂点数が 10000 以上あるような大規模ネットワークでは, 既存のアルゴリズムよりも 1000 倍以上高速化されている.

また, 実験 2 より, 提案したアルゴリズムの解 1 つあたりの計算時間は, ほぼ $O(\Delta^2)$ であることが確認される. 極大クリーク, あるいは極大 2 部クリークを 1 つ求めるには, 貪欲法が高速であり, その計算時間は, 前処理の時間を除くと $O(\Delta^2)$ である. 興味深いことに, これらの時間計算量は等しく, 列挙が, 貪欲法で多量のクリークをヒューリスティックに求めるアルゴリズムと比べても, 計算時間のオーダー的には遜色がないことが分かる.

実験 3 より, 定数個の頂点の次数が大きくても, 計算時間はそれ以外の頂点の最大次数にしか依存しないことも分かる. なお, 実験 3 の結果で, 頂点数が 32000 以上のものについて計算時間が増えているのは, 使用するメモリが PC のメモリより大きくなり, スワップが発生しているため, CPU の 1 次・2 次キャッシュのヒット率が極端に低下したことに起因すると推測される.

実験 4 で、問題 1、問題 2 ともに短時間で列挙ができていることから、現実問題のデータでも、本稿のアルゴリズムは高速に動作することがある程度確認された。頂点の添え字を次数順でソートした場合としない場合では計算時間に 50 倍程度の開きがあり、次数にばらつきがある場合には、ソートすることによる計算時間短縮の効果が大きいことが確認された。

7. ま と め

本研究では, 巨大で疎なグラフの極大なクリークと極大な 2 部クリークを列挙するアルゴリズムを提案した. グラフの最大次数を Δ とすると, 本稿の改良により, 極大クリーク 1 つあたりの計算時間が $O(|V||E|)$ から $O(\Delta^4)$ に, 極大 2 部クリークは $O(|V||E|)$ から $O(\Delta^3)$ に改善された. また, 計算機実験により, このアルゴリズムのランダムな入力に対する計算時間は $O(\Delta^2)$ 程度であることが確認された. また, 頂点の次数にばらつきがあるグラフに対しても, 同様な時間計算量が得られ, 実際の計算も高速であることが確認された. また、現実のデータに対しても、これらの結果と同じパフォーマンスを示すことを確認した。

文　献

[1] R. Agrawal and R. Srikant, “Fast Algorithms for Mining Association Rules in Large Databases,” *In Proceedings of VLDB ’94,* pp. 487–499, 1994.

[2] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen and A. I. Verkamo, “Fast Discovery of Association Rules,” *In Advances in Knowledge Discovery and Data Mining,* MIT Press, pp. 307–328, 1996.

[3] D. Avis and K. Fukuda, “Reverse Search for Enumeration,” *Discrete Applied Mathematics,* Vol. 65, pp. 21–46, 1996.

[4] S. R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins, “Trawling the Web for Emerging Cyber-Communities,” *Proceedings of the Eighth International World Wide Web Conference,* Toronto, Canada, 1999.

[5] S. Tsukiyama, M. Ide, H. Ariyoshi and I. Shirakawa, “A New Algorithm for Generating All the Maximum Independent Sets,” *SIAM Journal on Computing,* Vol. 6, pp. 505–517, 1977.